**Blueprint del Sistema de Algo Trading**

**1) Estructura del repositorio**

algo-trading/

├─ README.md

├─ .env.example

├─ pyproject.toml / requirements.txt

├─ docker-compose.yml

├─ Makefile

├─ configs/

│ ├─ symbols.yml # universo de activos e intervalos (BTCUSDT: [1d,4h], ...)

│ ├─ data\_paths.yml # rutas del Data Lake (raw/trusted/curated)

│ ├─ risk.yml # límites: dd diario, sizing, SL/TP por estrategia

│ ├─ strategies.yml # estrategias activas y parámetros

│ └─ orchestration.yml # ventanas horarias, SLAs, alertas

├─ data/ # Data Lake local (Parquet); versionado en cloud (B2/S3)

│ ├─ raw/ # JSON/API tal cual (opcional, para auditoría)

│ ├─ trusted/ # validado (tipos + rangos OHLCV)

│ └─ curated/ # listo para consumo (OHLCV final, particionado)

├─ infra/

│ ├─ docker/ # definiciones de servicios

│ ├─ prefect/ # despliegues/infra del orquestador Prefect

│ └─ grafana/ # dashboards, alert rules (si aplicas)

├─ services/

│ ├─ api/ # FastAPI (punto único de entrada del sistema)

│ │ ├─ main.py # endpoints: /ingest, /trade, /health, /kill\_all

│ │ ├─ deps.py # carga de configs, clientes (DuckDB, Postgres, Binance)

│ │ └─ schemas.py # modelos Pydantic (payloads y respuestas)

│ ├─ ingestion/ # ingestores REST/WS → Parquet

│ │ ├─ binance\_client.py # llamadas REST klines + utilidades (rate-limit/backoff)

│ │ ├─ writer.py # upsert Parquet (particiones y de-dup por índice)

│ │ └─ pipeline.py # “download→validate→write→checkpoint”

│ ├─ processing/ # validación, resample, feature engineering

│ │ ├─ validate.py # checks OHLCV, UTC, duplicados, huecos

│ │ ├─ features.py # RSI, MACD, MAs, ATR... (stateless)

│ │ └─ dataset.py # construcción de datasets para modelos

│ ├─ models/ # ML/DL + reglas determinísticas

│ │ ├─ rules/ # estrategias por reglas (EMA crossover, MR, momentum…)

│ │ ├─ ml/ # sklearn/xgboost; pipelines, scaler

│ │ ├─ dl/ # torch/tf; LSTM/Transformer

│ │ └─ registry.py # interfaz común: .fit() .predict() .signal()

│ ├─ backtest/ # motor de backtesting event-driven

│ │ ├─ engine.py # simulador de fills, fees, slippage, latencia

│ │ ├─ costs.py # maker/taker, funding, borrow, impacto

│ │ ├─ metrics.py # sharpe, sortino, maxDD, hit-rate, PF

│ │ └─ reports.py # resúmenes, atribución PnL, gráficos (si aplicas)

│ ├─ execution/ # OMS/EMS (Binance)

│ │ ├─ binance\_exec.py # spot/futures: órdenes market/limit/stop/TP, reduceOnly

│ │ ├─ risk.py # sizing, límites, kill-switch

│ │ └─ reconciler.py # estado de órdenes/posiciones, idempotencia

│ ├─ orchestration/ # Flujos Prefect (jobs diarios/5-min)

│ │ ├─ flows\_ingest.py # ingesta incremental (1d, 4h)

│ │ ├─ flows\_risk.py # health-check, umbrales, kill-switch

│ │ └─ flows\_analysis.py # tareas periódicas de features/validaciones

│ ├─ n8n/ # flujos exportados (JSON) para importarlos en n8n

│ │ └─ algo\_orchestration\_n8n.json

│ └─ monitoring/ # logs y métricas (formato, exportadores)

│ ├─ logging.json # estructura de logs (orden, fill, error, alerta)

│ └─ metrics.md # métricas expuestas /health y Prometheus (si aplicas)

├─ tests/ # tests unitarios/integración (pytest)

│ ├─ test\_ingestion.py

│ ├─ test\_features.py

│ ├─ test\_backtest\_engine.py

│ └─ test\_execution.py

└─ tools/

├─ duckdb\_queries/ # consultas SQL útiles para exploración

└─ scripts/ # utilidades (backfill puntual, “fix gaps”, etc.)

**2) Convenciones y configuración**

* Variables de entorno (archivo .env):

BINANCE\_API\_KEY, BINANCE\_API\_SECRET, MODE (testnet|prod), DATA\_ROOT, DB\_URL (opcional Postgres), TELEGRAM\_TOKEN/CHAT\_ID (si usas alertas).

* Config central con YAML (configs/):

• symbols.yml: lista de símbolos e intervalos; • risk.yml: límites por estrategia; • strategies.yml: estrategias activas; • orchestration.yml: SLAs, horarios y umbrales.

* Tiempos y zonas: **siempre UTC**.
* Idempotencia: toda operación “externa” debe tener un clientOrderId único y reintentos safe.

**3) Data Lake (Parquet) y esquema**

* Estructura de particiones en curated/:

symbol=SYMB/interval=1d/year=YYYY/month=MM/data.parquet

symbol=SYMB/interval=4h/year=YYYY/month=MM/data.parquet

* Columnas base: date (índice), open, high, low, close, volume, number\_of\_trades, quote\_asset\_volume.
* Validaciones mínimas:

low ≤ min(open, close, high); high ≥ max(open, close, low); volumen ≥ 0; sin duplicados por (symbol, interval, date).

* Checkpoint por (symbol|interval): último close\_time almacenado, para ingestas incrementales diarias/4h.

**4) Ingesta: responsabilidades e interfaces**

* services/ingestion/binance\_client.py

Funciones de lectura REST para klines (1d, 4h) con backoff y respeto de rate-limits.

* services/ingestion/writer.py

Upsert a Parquet: concatena partición existente + nuevas filas, ordena por date y de-dup por índice.

* services/ingestion/pipeline.py

Orquesta: lee checkpoint → descarga delta → valida → escribe trusted/curated → actualiza checkpoint → emite evento/metric.

Interfaces sugeridas (nombres orientativos):

• fetch\_klines(symbol, interval, start\_ms, end\_ms=None) -> list[rows]

• validate\_frame(df) -> df\_valid

• upsert\_parquet(df, path, partition\_keys) -> stats

• update\_checkpoint(key, last\_close\_time)

**5) Procesamiento y features**

* services/processing/validate.py

Limpieza y validación; normalización de tipos; tolerancias de huecos.

* services/processing/features.py

Funciones puras (stateless): rsi(df, period), ema(df, span), atr(df, n), etc.

* services/processing/dataset.py

Construcción de datasets por ventana temporal y horizonte objetivo (clasificación o regresión).

Interfaz:

• build\_dataset(symbol, interval, features\_cfg, horizon, target) -> DataFrame

**6) Modelos y reglas**

* Estrategias por reglas (simple y robusto para empezar):

ema\_crossover(fast, slow), mean\_reversion(rsi\_low, rsi\_high), filtros de volatilidad/volumen.

* ML/DL (incremental):

Pipelines sklearn (scalers, CV) y LSTM/Transformer para series.

* Registro de modelos:

registry.py expone .fit(df), .predict(df), .signal(latest\_row) de forma uniforme (para intercambiar la estrategia sin tocar la ejecución).

Señales normalizadas (objeto simple):

Signal = { "symbol": "BTCUSDT", "side": "BUY|SELL|FLAT", "conf": 0.0..1.0, "meta": {...} }

**7) Backtesting (event-driven realista)**

* services/backtest/engine.py

Simula órdenes y fills con: fees maker/taker, slippage dependiente de volumen y volatilidad, latencias, parciales, rechazos.

* services/backtest/costs.py

Modelo de costes del exchange (comisiones, funding si perps).

* services/backtest/metrics.py

KPIs: CAGR, Sharpe, Sortino, MaxDD, HitRate, ProfitFactor, Exposure, Turnover.

* services/backtest/reports.py

Resumen de resultados y atribución (alpha, costos, timing).

Interfaz:

• run\_backtest(strategy, data, costs, risk\_cfg) -> BacktestResult

• BacktestResult con equity curve, trades, métricas y breakdown de costes.

**8) Ejecución en vivo (OMS/EMS)**

* services/execution/binance\_exec.py

Clientes para Spot y USDT-M Futures (modo one-way/hedge):

abrir/cerrar posición, órdenes market/limit, stop, take-profit, OCO (spot), reduceOnly (futuros).

* services/execution/risk.py

Sizing por volatilidad objetivo; SL/TP por diseño; límites de exposición por símbolo y DD diario (kill switch).

* services/execution/reconciler.py

Reconciliación: confirma fills, evita duplicados (idempotencia), sincroniza estado interno con el exchange.

Interfaz:

• submit\_order(signal, sizing\_ctx) -> ExecReport

• place\_stop\_tp(position, sl, tp)

• close\_all\_positions() (kill-switch)

**9) API del sistema (FastAPI)**

* Endpoints esenciales:

• POST /ingest  → cuerpo: {symbol, interval}; baja delta y actualiza Parquet.

• POST /trade   → cuerpo: {symbol, side, qty, sl?, tp?, mode: spot|futures}; ejecuta orden conforme a risk.yml.

• GET  /health  → retorna métricas (dd, pnl\_day, positions\_open, latency\_ms, last\_ingest\_ts, last\_trade\_ts, rate\_limit\_hits).

• POST /kill\_all → cancela órdenes y cierra posiciones, registra evento crítico.

* Seguridad:

API Key simple (header), allowlist IP (si corre expuesta), logs estructurados, límites de frecuencia.

**10) Orquestación y agentes**

* services/orchestration/flows\_ingest.py (Prefect):

job diario 00:05 UTC para 1d y job de 4h cada cierre de vela; usa checkpoint, valida y escribe Parquet.

* services/orchestration/flows\_risk.py (Prefect):

health-check cada 5 min; si dd ≥ umbral → POST /kill\_all y alerta.

* n8n (services/n8n/\*.json):

• Webhook /signal para señales externas (TradingView) → POST /trade.

• Cron para disparar análisis o resúmenes de noticias (si se integra OpenAI).

• Alertas Telegram/Discord con resultados y fallos.

**11) Observabilidad**

* Logs estructurados (JSON line):

event=order\_submitted | order\_filled | ingest\_ok | health\_alert | kill\_switch, con campos: ts, symbol, side, qty, price, latency\_ms, err, trace\_id.

* Métricas:

expuestas en /health o en un endpoint /metrics (si usas Prometheus): dd, pnl\_day, orders\_sent, orders\_failed, ingest\_latency, rate\_limit\_hits.

* Dashboards:

Grafana/Metabase para equity curve, PnL diario, drawdown, conteo de trades, latencias.

**12) Seguridad y resiliencia**

* Claves Binance: permisos mínimos; en prod, IP allowlist; rotación periódica.
* NTP activo (sincronía de reloj).
* Idempotencia en órdenes (clientOrderId) y reintentos con backoff.
* Kill-switch manual y automático (umbral de DD).
* Testnet primero; rollout gradual (1% → 5% → 20% del capital).

**13) Testing**

* Unit tests:

• ingestion: validación de OHLCV y de-dup;

• features: precisión de indicadores;

• execution: construcción de órdenes con reduceOnly y SL/TP;

• backtest: métricas consistentes con escenarios conocidos.

* Integration tests:

• /ingest con fake API;

• /trade con binance sandbox;

• flujo Prefect + n8n (happy path + fallos simulados).

* Simulaciones de fracaso:

desconexión, throttling (HTTP 429), error de firma, latencia alta.

**14) Roadmap (prioridades de implementación)**

1. Ingesta incremental (1d, 4h) → Parquet curated + checkpoint.
2. API FastAPI con /ingest y /health; orquestación Prefect de ingesta.
3. Validaciones + features básicas; DuckDB queries útiles.
4. Estrategias por reglas (EMA/RSI) + motor de backtest; reportes y métricas.
5. /trade + ejecución Binance testnet (SL/TP, reduceOnly) + reconciliación.
6. Monitoreo (logs, /health completo), kill-switch y alertas n8n.
7. Experimentos ML simples e integración opcional de noticias/OpenAI.
8. Postgres/Timescale (si necesitas servir datos a dashboards externos).
9. Endurecimiento (CI/CD, pruebas de carga, canary, backups).

**15) Estándares de entrega (DoD por módulo)**

* Ingesta: idempotente, valida OHLCV, escribe Parquet particionado, actualiza checkpoint, registra stats.
* Features: funciones puras, deterministas, testeadas con tolerancias.
* Backtest: reproduce resultados con semilla fija; reporta KPIs y breakdown de costes.
* Ejecución: idempotencia, SL/TP automáticos, manejo de errores y reconexiones, logging completo.
* API: contratos definidos (schemas), errores claros, autenticación, tests.
* Orquestación: retries configurados, SLAs definidos, alertas en fallback.
* Observabilidad: logs JSON, /health con métricas, panel básico.

**16) Qué programar primero (mini-guía)**

* services/ingestion/pipeline.py: “descarga→valida→escribe→checkpoint” para 1d y 4h.
* services/api/main.py: /ingest (usa pipeline) y /health (exponer métricas base).
* services/orchestration/flows\_ingest.py: Prefect para correr diario y cada 4h.
* services/processing/features.py: EMA/RSI/ATR.
* services/backtest/engine.py: motor sencillo con comisiones y slippage fijo.
* services/execution/binance\_exec.py: órdenes market + SL/TP (testnet) con idempotencia.
* services/orchestration/flows\_risk.py: health guard y kill-switch.
* n8n: webhook /signal → POST /trade; alertas Telegram de éxito/fallo.